REMARKS

Claims 1, 18, 34, 35, and 52 were amended. Claims 1-9, 11-27, 29-43, 45-60 and 62-68 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 103(a) Rejections:

The Examiner rejected claims 1-8, 10-13, 16, 34-42, 45-47, 50, 52-59, 62-64 and 67 under 35 U.S.C. § 103(a) as being unpatentable over Monday et al. (U.S. Patent 6,263,377) (hereinafter "Monday") in view of Venners ("Inside the Java Virtual Machine"), and claims 9, 14, 15, 17-27, 29-33, 43, 48, 49, 51, 60, 65, 66 and 68 as being unpatentable over Monday in view of Venners and further in view of Babaoglu et al. ("Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems") (hereinafter "Babaoglu"). Applicants respectfully traverse these rejections for at least the following reasons.

Contrary to the Examiner's assertion, Monday in view of Venners fails to teach or suggest a remote class loader mechanism configured to: detect the indication that the class is not loaded; obtain the class from a remote system via a network; and store the class in a location indicated by the class path of the default class loader on the system; wherein the remote class loader mechanism is configured to perform said detect, said obtain, and said store separate from and transparent to the default class loader. In claim 1 of the present application, *detecting the indication that the class is not loaded,* said *obtaining the class from a remote system via a network,* and said *storing the class in a location on the system indicated by the class path of the default class loader means* are all performed separately from and transparently to the default class loader for the virtual machine. The default class loader attempts to load a class from a class path of the default class loader means. If the class is not located, the recited remote class loader detects the indication that the class needed to execute the code on the system is not stored in the one or more locations indicated by the class path,

obtains the class from a remote system via a network, and stores the class in a location on the system indicated by the class path of the default class loader means. These operations are separate from and transparent to the default class loader. The default class loader may then load the class from the location on the system indicated by the class path.

Page 1, line 10-page 3, line 5 of the Background section of the instant application generally describes the dynamic loading of classes using class loaders in the prior art. In page 1 line 28 – page 2 line 12, Applicants describe the concept and general operations of custom class loaders in virtual machines. The first part of this section states:

> Virtual machines such as JVM may provide a facility by which a user can introduce a custom class loader. For example, in JVM, <u>a hook is provided to the loading mechanism through the custom class loaders</u>. Programmatically speaking, [custom] class loaders are ordinary objects that may be defined in code (e.g. Java™ code). In Java™, [custom] <u>class loaders are instances of subclasses of abstract class Classloader</u>.

A conventional mechanism for dynamically loading classes not available on the class path of the default class loader for a virtual machine is via custom class loaders, as is described on page 3, lines 4-5 of the present application:

> In Java, to use a class, the class has to be in the class path of the default class loader, or alternatively a custom class loader may be provided.

Claim 1 of the instant application recites a remote class loader mechanism to remotely load classes needed to run, for example, an application in a distributed computing environment through the <u>default</u> class loader of the virtual machine (i.e., <u>not</u> a <u>custom</u> class loader(s) to remotely load the classes) (see, for example, page 3, lines 9-11 and page 4, lines 3-11 of the present application).

In contrast, what the Monday reference describes in col. 3, lines 38-56 and elsewhere is the use of a <u>custom</u> class loader as disclosed in the background section of the present application. In the cited section, Monday states "If x.class is NOT located, then a <u>**subclass of**</u> <u>the CLASSLOADER</u>, a REMOTECLASSLOADER…" In the background section of the present application, as noted above, Applicants explain that <u>custom class loaders are instances of **subclasses** of abstract class Classloader</u>. When Monday refers to

REMOTECLASSLOADER, Monday is referring to a custom class loader. Monday's system uses a custom class loader in the conventional method as described in the background section of the present application to load classes not found on the class path of Monday's CLASSLOADER.

Thus, what Monday discloses is clearly and distinctly different than what is recited in claim 1 of the present application. Monday's system is actually described as operating according to the conventional method for loading classes that relies on custom class loaders. Monday's REMOTECLASSLOADER is simply a custom class loader as described in Applicants' background section. Applicants' specification discloses a method for loading classes that intentionally does not rely on the conventional method in virtual machines using custom class loaders to avoid problems created thereby; instead, Applicants' disclosed method transparently determines or detects that a class has not been loaded from a location on the class path of the default class loader, locates the class, obtains the class, and stores the class in the location indicated by the class path of the default class loader *without* using custom class loaders. The default class loader may then load the class from the location indicated by the class path of the default class loader. This is the system recited in claim 1.

In summary, Monday describes a method that relies on a custom class loader, while claim 1 recites a system for implementing a method for remotely loading classes that does not rely on a custom class loader. Also, Monday does not teach detecting an indication that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path, means for obtaining the class from a remote system via a network, and storing the class in a location on the system indicated by the class path of the default class loader are all performed separately from and transparently to the default class loader for the virtual machine. **By definition, since Monday's REMOTECLASSLOADER is a subclass of the CLASSLOADER, it cannot and does not operate separately from and transparently to the default CLASSLOADER.** Clearly, Monday does not anticipate claim 1 of the instant application.

Applicants note that the Examiner asserts, with respect to this limitation, "inherent from the remote class loader check the remoteclasspath, obtain the class and store it in the directory without consulting from the class loader; col. 2, lines 44-56". **The Examiner's assertion is entirely unsupported by the actual teachings of the reference**. Instead, Applicant submits that the remoteclassloader is likely called by the classloader in the event that the class is not found in the classpath. Applicant respectfully submits that it is clearly not inherent that Monday's remoteclassloader operates <u>both</u> separately from <u>and</u> transparently with respect to the classloader. As the Examiner is certainly aware, according to M.P.E.P. 2131.01 III, in regard to a theory of inherency "evidence <u>must</u> make clear that the missing descriptive matter <u>is **necessarily** present</u> in the thing described in the reference, and that it would be so recognized by persons of ordinary skill." The Examiner has not provided anything but the Examiner's own unsupported speculation that the functionality relied upon by the Examiner is inherent in Monday. The rejection is not supported by the actual evidence of record.

Furthermore, Monday in view of Venners fails to teach or suggest wherein the default class loader is independent from the remote class loader mechanism. As already indicated above, the remoteclassloader (which the Examiner currently relies on as the remote class loader) of Monday is a subclass of classloader. Thus, **by definition,** the Monday's classloader is <u>not</u> independent from the remoteclassloader.

In regards to Venners, page 1, line 10-page 3, line 5 of the Background section of the instant application generally describes the dynamic loading of classes using class loaders in the prior art. In particular, page 3, lines 4-5 describes that:

> In Java, to use a class, the class <u>has to be in the class path of the default class loader, or alternatively a **custom** class loader may be provided</u>.

The Venners reference is an overview of the Java Virtual Machine (JVM). Chapters 1-4 "give a broad overview of Java's architecture". A careful review of the Venners reference, particularly an overview given in Chapter 3, page 2, shows that Venners' description of Java's method of loading classes using class loaders <u>is consistent with what is described in the Background section of the instant application</u>. For example,

Venners, in Chapter 3, page 2, gives the following description of how Java's class loading mechanism works:

> Imagine that during the course of running the Java application, a request is made of your [custom] class loader to load a class named Volcano. Your [custom] class loader would first ask its parent, the **class path class loader**, to find and load the class. The **class path class loader**, in turn, would make the same request of its parent, the installed extensions class loader. This class loader, would also first delegate the request to its parent, the bootstrap class loader. Assuming that class Volcano is not a part of the Java API, an installed extension, or **on the class path**, all of these class loaders would return without supplying a loaded class named Volcano. When the **class path class loader** responds that neither it nor any of its parents can load the class [i.e., the requested class is **not on a class path**], your [custom] class loader could then attempt to load the Volcano class in its **custom** manner, by downloading it across the network. Assuming your [custom] class loader was able to download class Volcano, that Volcano class could then play a role in the application's future course of execution.

In contrast to the above description from Venners, claim 1 of the instant application recites that a <u>*default class loader for a virtual machine*</u> determines that a class needed to execute code on the system is not stored in one or more local locations indicated by the class path (and thus fails to load the class). An indication is generated that the class is not loaded. A <u>remote class loader mechanism</u> then, transparently to the default class loader of the virtual machine, detects the indication, obtains the class from a remote system via a network, and stores the class in a location indicated by the class path. The <u>default class loader</u> (**not** a <u>custom class loader</u>, as in the Venners reference and the Monday reference, and as the Java class loading mechanism described by Venners and described in the Background section of the instant application operates) <u>then loads the class from the location indicated by the class path</u>.

To summarize, Venners discloses that, if the "default class loader" (**class path class loader**) cannot locate a class in a location on its class path, the "default class loader" <u>notifies</u> a "custom class loader" associated with the class, which then attempts to load the class, possibly from a remote location. Similarly, the Monday reference describes, in col. 3, lines 38-56, a REMOTECLASSLOADER that clearly operates as a

conventional custom class loader as is disclosed in Venners and in the background section of the present application.

In contrast, claim 1 of the instant application recites that, if the "default class loader" cannot locate a class in a location on its class path, a "remote class loader mechanism", transparently to the default class loader, determines that the class was not located on the class path of the default class loader, locates the class on a remote system, and stores the class in a location indicated by the default class loader's class path. The default class loader then loads the class from the location.

Thus, what claim 1 of the instant application recites is clearly distinct from the Java class loading mechanism as described by the Venners reference and from Monday's disclosed system.

Additionally, Monday in view of Venners fails to teach or suggest wherein the default class loader being configured to load the class from the location avoids class conflicts. Monday nowhere discloses this feature. There is no mention in either Monday or Venners as to avoiding class conflicts when loading from a particular location. Furthermore, as already indicated above, Venners teaches the use of custom class loaders for loading remote classes which Applicants are specifically avoiding in claim 1. As indicated in the background of the present application, using custom class loaders can lead to class conflicts. Accordingly, the cited art does not teach this feature of claim 1.

Applicants remind the Examiner that, to establish a *prima facie* case of obviousness of a claimed invention, all claim limitations must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 U.S.P.Q. 580 (C.C.P.A. 1974), MPEP 2143.03. As shown above, the cited art does not teach or suggest all limitations of claim 1.

Furthermore, the Examiner's stated motivation for combining the references, "because Venners teaches the details how a Java application load and utilize class that are needed at runtime", is clearly not a valid reason for combining the references. It is if anything simply a "motive" for understanding how the Monday system already works, since the Monday system already employs the conventional class loading method described in Venners. Furthermore, the present application describes the conventional method for dynamically loading classes in virtual machines such as JVMs as is described in Venners, and clearly discloses a system and method that is distinctly different than the conventional method described in Venners. Claim 1 of the instant application clearly recites elements that enable the distinctly different method described in Applicants' specification. Moreover, the Examiner's stated reason is merely conclusory.

Furthermore, even if the Monday and Venners references were combined, the combination would not produce anything like what is recited in claim 1 of the present application. It would simply produce a system that handles the loading of classes using the Java class loading method as disclosed in Venners, which it is clear that Monday's system already uses. In other words, it would simply produce the Monday system as disclosed in the Monday reference. **Since the Monday reference already employs the class loading mechanism disclosed in Venners, there is and can be no motivation to combine the references in any case.**

Thus, for at least the reasons presented above, the rejection of claim 1 is not supported by the cited prior art and removal thereof is respectfully requested. Similar remarks as those above regarding claim 1 also apply to claims 34, 35, and 52.

With regard to claim 17, contrary to the Examiner's assertion, the cited references fail to disclose, alone or in combination, program instructions executable by a processor to implement *wherein the system and the remote system are configured to participate in a distributed computing system on the network for submitting computational tasks in a distributed heterogeneous networked environment that utilizes*

peer groups to decentralize task dispatching and post-processing functions and enables a plurality of jobs to be managed and run simultaneously. The Examiner simply asserts that Babaoglu teaches "peer-to-peer application can be implemented in Java (page 7, section 4)" and that "it would have been obvious…to apply the teachings of Babaoglu to the system of Monday because it presents a framework supporting a new approach for building P2P application in which resource can be sharing by direct exchange between peer nodes."

The Babaoglu reference describes:

> …Anthill, a framework to support the design, implementation and evaluation of P2P applications based on ideas such as multi-agent and evolutionary programming borrowed from CAS. An Anthill system consists of a dynamic network of peer nodes; societies of adaptive agents travel through this network, interacting with nodes and cooperating with other agents in order to solve complex problems. (Babaoglu, Abstract).

However, the Examiner has simply asserted that "Babaoglu teaches "peer-to-peer application can be implemented in Java", and has not provided any argument or reference that Babaoglu teaches or suggests all of the claim limitations recited in claim 17 of the instant application. For example, the Examiner has provided no argument or reference, nor can the Applicants find anything, from the Babaoglu reference that teaches or suggests that the Babaoglu "Anthill" framework is a distributed computing system for submitting computational tasks in a distributed heterogeneous networked environment or that the Babaoglu "Anthill" framework *decentralizes task dispatching and post-processing functions* and *enables a plurality of jobs to be managed and run simultaneously*.

Applicants remind the Examiner that, to establish a *prima facie* case of obviousness of a claimed invention, all claim limitations must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 U.S.P.Q. 580 (C.C.P.A. 1974), MPEP 2143.03. As shown above, the cited art does not teach or suggest all limitations of claim 1, from which claim 17 depends.

Furthermore, the Examiner's stated motivation for combining the references, "because it presents a framework supporting a new approach for building P2P application in which resource can be sharing by direct exchange between peer nodes", is not a motivation that is directly relevant to Monday's method and computer program product for managing distributed applications on a local computer system. It is **not at all** "obvious" how Monday's disclosed method would benefit from Babaoglu "Anthill" framework, nor is it obvious as to how Babaoglu's "Anthill" framework could, or even if it could, be implemented in Monday's disclosed method. **Furthermore, the stated motivation is not a motivation that is directly relevant to the functioning or application of what is recited in claim 17 of the instant application.** Moreover, the Examiner's stated motivation is merely conclusory.

Furthermore, as shown above, even if the Monday and Babaoglu references were combined, the combination would not produce anything like was is recited in claim 17 of the present application.

Thus, for at least the reasons presented above, the rejection of claim 17 is not supported by the cited prior art and removal thereof is respectfully requested. Similar remarks as those above regarding claim 17 also apply to claim 51 and 68.

Regarding independent claim 18, Applicants respectfully traverse this rejection for at least the reasons given above in regards to claim 1 and claim 17.

Applicants note that the Examiner has not addressed the arguments from above regarding claims 17, 18, 51, and 68.

Applicants also assert that the rejection of numerous ones of the dependent claims is further unsupported by the cited art. However, since the rejection has been shown to be unsupported for the independent claims, a further discussion of the dependent claims is not necessary at this time.

CONCLUSION

Applicants submit the application is in condition for allowance, and notice to that effect is respectfully requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5681-65900/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicants

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: February 28, 2008